GOVERNMENT COLLEGE OF ENGINEERING

ERODE-638316

RECORD NOTE BOOK

REGISTER NUMBER | |

Certified that this is the Bonafide record of work done by Selvan / Selvi ……. of the **Fourth** Semester **Computer Science and Engineering** Branch during the academic year **2022-2023** in the **Artificial Intelligence and Machine Learning(CS3491)** Laboratory.

STAFF INCHARGE **HEAD OF THE DEPARTMENT**

Submitted for the university practical examination on ……. at Government College of Engineering.

Date……..

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

Ex.No:1 **Implementation of Uninformed Search Algorithm**
Date:

Aim:

To write a python program to implement Uninformed Search Algorithms such as Breadth-First Search and Depth First Search strategies.

Program:

a) Breadth First Search

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [ ] # List for visited nodes.
queue = [ ]     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling
```

b) Depth First Search

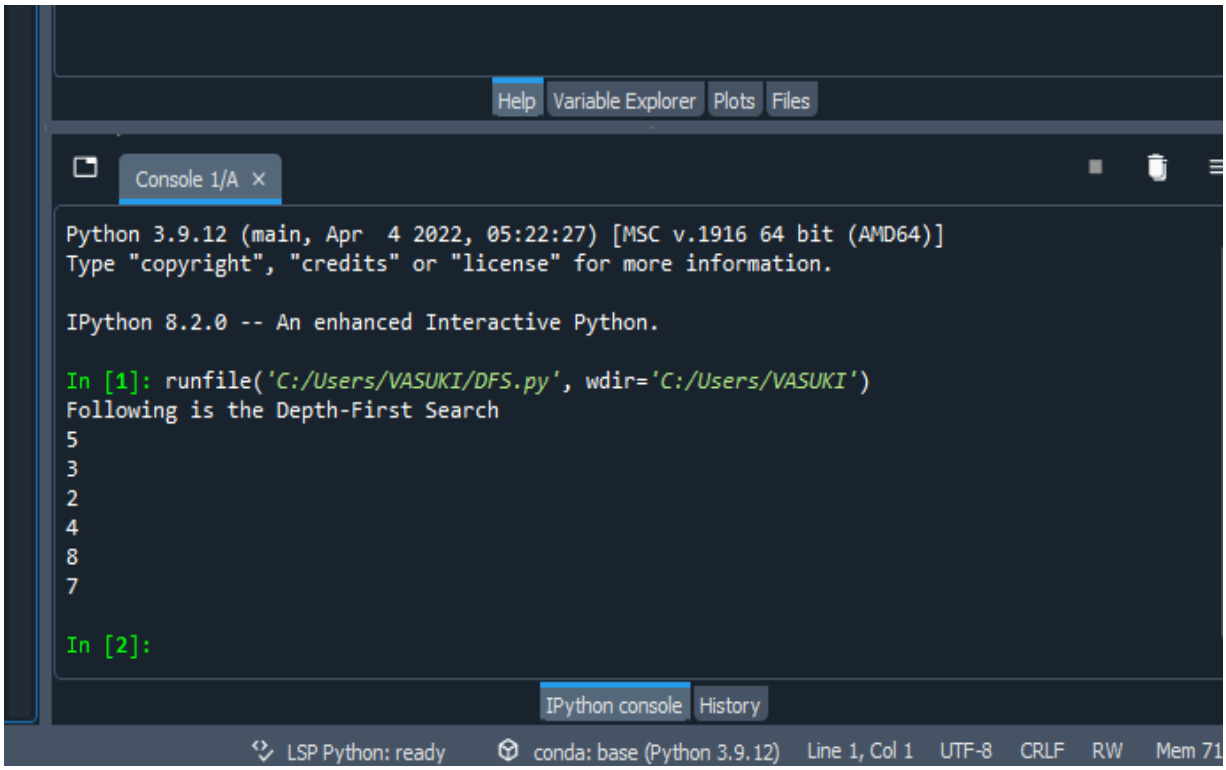
```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
```

```
visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

A. Breadth First Search

B.Depth First Search



Result:

Thus the python program to implement uninformed search strategies was executed and output was verified successfully.

Ex.No:2 **Implementation of Informed Search Algorithm**

Date:

Aim:

To write a python program to implement Informed Search Algorithms such as A* algorithm and SMA* algorithm.

Program:

a) A* Algorithm

```
from copy import deepcopy
import numpy as np
import time

def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count][&#39;puzzle&#39;], 0)
        count = (state[count][&#39;parent&#39;])
    return bestsol.reshape(-1, 3, 3)

# checks for the uniqueness of the iteration(it).
def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
        else:
            return 0

# number of misplaced tiles
def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost &gt; 0 else 0

def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
        pos[q] = p
    return pos

# start of 8 puzzle evaluvation, using Misplaced tiles heuristics
```

```
def evaluvate_misplaced(puzzle, goal):
    steps = np.array([(&#39;up&#39;, [0, 1, 2], -3),(&#39;down&#39;, [6, 7, 8],
    3),(&#39;left&#39;, [0, 3, 6], -1),(&#39;right&#39;, [2, 5, 8],  1)],
    dtype =  [(&#39;move&#39;,  str, 1),(&#39;position&#39;, list),(&#39;head&#39;,
   int)])
    dtstate = [(&#39;puzzle&#39;,  list),(&#39;parent&#39;,
   int),(&#39;gn&#39;,  int),(&#39;hn&#39;,  int)]
    costg = coordinates(goal)
    parent = -1
    gn = 0
    hn = misplaced_tiles(coordinates(puzzle), costg)
    state = np.array([(puzzle, parent, gn, hn)], dtstate)
    dtpriority = [(&#39;position&#39;, int),(&#39;fn&#39;, int)]
     priority = np.array([(0, hn)], dtpriority)

    while 1:
       priority = np.sort(priority, kind=&#39;mergesort&#39;, order=[&#39;fn&#39;,
      &#39;position&#39;])
       position, fn = priority[0]
       priority = np.delete(priority, 0, 0)
       puzzle, parent, gn, hn = state[position]
       puzzle = np.array(puzzle)
      blank = int(np.where(puzzle == 0)[0])
      gn = gn + 1
       c = 1
       start_time = time.time()
       for s in steps:
          c = c + 1
          if blank not in s[&#39;position&#39;]:
             openstates = deepcopy(puzzle)
             openstates[blank], openstates[blank + s[&#39;head&#39;]] =
              openstates[blank + s[&#39;head&#39;]], openstates[blank]

             if ~(np.all(list(state[&#39;puzzle&#39;]) == openstates,
              1)).any():
                end_time = time.time()
                if (( end_time - start_time ) &gt; 2):
                   print(&quot; The 8 puzzle is unsolvable \n&quot;)
                   break
                hn = misplaced_tiles(coordinates(openstates), costg)
                q = np.array([(openstates, position, gn, hn)],
               dtstate)
               state = np.append(state, q, 0)
                # f(n) is the sum of cost to reach node
                fn = gn + hn
                q = np.array([(len(state) - 1, fn)], dtpriority)
```

```
                priority = np.append(priority, q, 0)
                if np.array_equal(openstates,goal):
                    print(&#39; The 8 puzzle is solvable \n&#39;)
                    return state, len(priority)

    return state, len(priority)
# initial state
puzzle = []
puzzle.append(2)
puzzle.append(8)
puzzle.append(3)
puzzle.append(1)
puzzle.append(6)
puzzle.append(4)
puzzle.append(7)
puzzle.append(0)
puzzle.append(5)
#goal state
goal = []
goal.append(1)
goal.append(2)
goal.append(3)
goal.append(8)
goal.append(0)
goal.append(4)
goal.append(7)
goal.append(6)
goal.append(5)
state, visited = evaluvate_misplaced(puzzle, goal)
bestpath = bestsolution(state)
print(str(bestpath).replace(&#39;[&#39;, &#39; &#39;).replace(&#39;]&#39;, &#39;&#39;))
totalmoves = len(bestpath) - 1
print(&#39;\nSteps to reach goal:&#39;,totalmoves)
visit = len(state) - visited
print(&#39;Total nodes visited: &#39;,visit, &quot;\n&quot;)
```

b)SMA* Algorithm

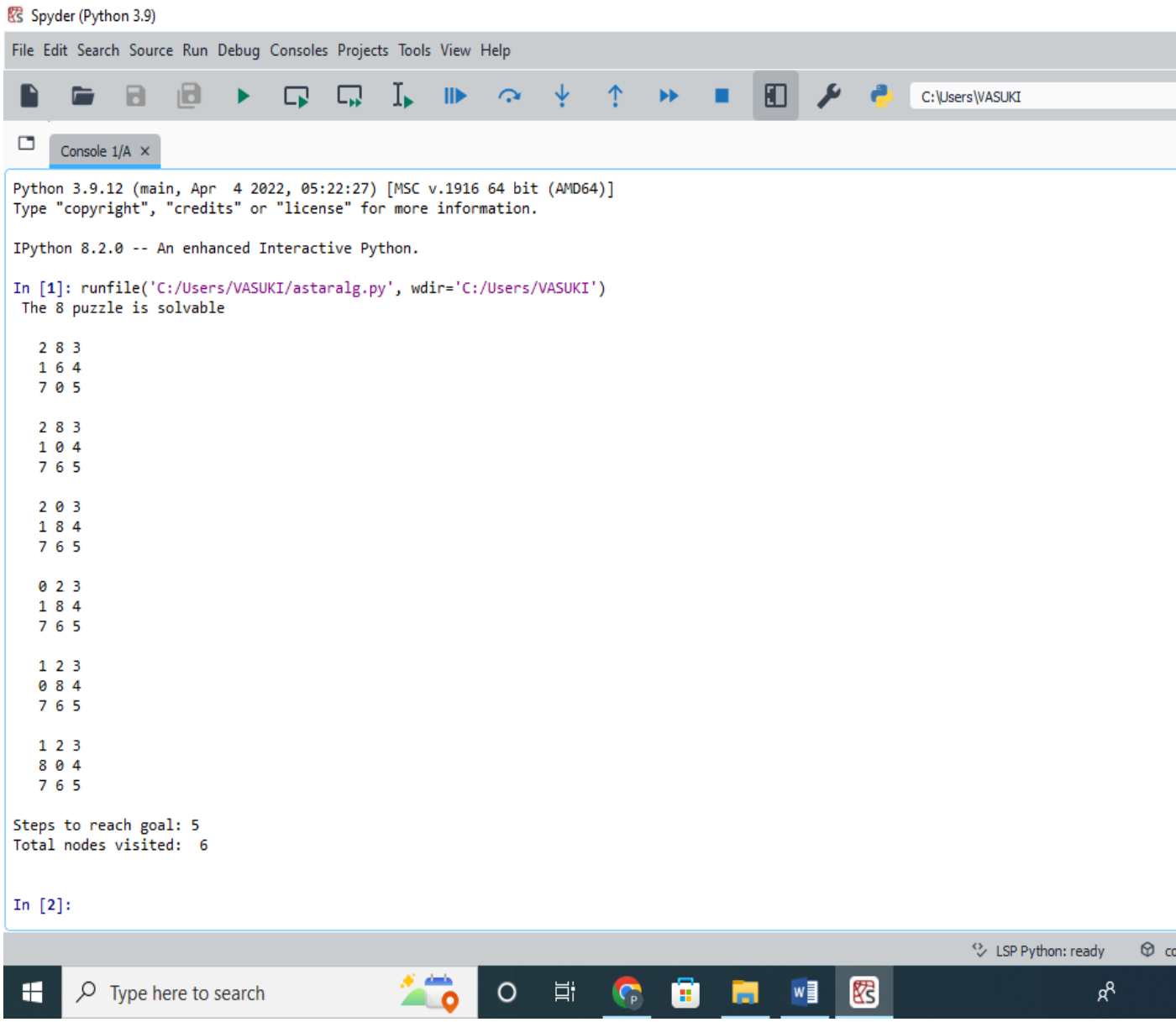
```
import heapq



class Node:

    def __init__(self, state, parent=None, g=0, h=0):

        self.state = state
```

```
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f


def calculate_heuristic(state, goal):
    # Replace this with an appropriate heuristic calculation for your problem
    return 0


def expand_node(node, graph):
    # Expand a node by generating its successor nodes
    successors = []
    for successor_state, step_cost in graph[node.state].items():
        g = node.g + step_cost
        h = calculate_heuristic(successor_state, goal_state)
        successor_node = Node(successor_state, node, g, h)
        successors.append(successor_node)
    return successors


def sma_star(start_state, goal_state, graph):
    start_node = Node(start_state)
    start_node.h = calculate_heuristic(start_state, goal_state)
    start_node.f = start_node.h
```

```
OPEN = [start_node]
CLOSED = []

while OPEN:
    current_node = heapq.heappop(OPEN)

    if current_node.state == goal_state:
        # Path found, return the solution
        path = []
        while current_node:
            path.append(current_node.state)
            current_node = current_node.parent
        return path[::-1]

    CLOSED.append(current_node)

    successors = expand_node(current_node, graph)

    for successor in successors:
        if successor in CLOSED:
            continue
        if successor in OPEN:
            existing_node = OPEN[OPEN.index(successor)]
            if successor.g < existing_node.g:
                existing_node.g = successor.g
                existing_node.parent = successor.parent
                existing_node.f = successor.f
        else:
            heapq.heappush(OPEN, successor)
```

```
    return None  # No path found


# Define the graph of Romania with step costs in kilometers
graph = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu': 80},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Rimnicu': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},
    'Pitesti': {'Rimnicu': 97, 'Craiova': 138, 'Bucharest': 101},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101}
}

start_state = 'Arad'
goal_state = 'Bucharest'

path = sma_star(start_state, goal_state, graph)

if path:
    print("Path found:", path)
else:
    print("No path found.")
```

A* Algorithm:



SMA* Algorithm

```
Path found: ['Arad', 'Sibiu', 'Rimnicu', 'Pitesti', 'Bucharest']
```

Result:

Thus the python program to implement informed search strategies was executed and output was verified successfully.

Ex.No:3 **Implementation of Naïve Bayes Model**
Date:

Aim:

To write a python program to implement Naïve Bayes Model.

Program:

```
# Make Predictions with Naive Bayes On The Iris Dataset
from csv import reader
from math import sqrt
from math import exp
from math import pi

# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
	for row in dataset:
		row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
	class_values = [row[column] for row in dataset]
	unique = set(class_values)
	lookup = dict()
	for i, value in enumerate(unique):
		lookup[value] = i
		print('[%s] => %d' % (value, i))
	for row in dataset:
		row[column] = lookup[row[column]]
	return lookup

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
	separated = dict()
	for i in range(len(dataset)):
```

```
                vector = dataset[i]
                class_value = vector[-1]
                if (class_value not in separated):
                        separated[class_value] = list()
                separated[class_value].append(vector)
        return separated

# Calculate the mean of a list of numbers
def mean(numbers):
        return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
        avg = mean(numbers)
        variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
        return sqrt(variance)

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
        summaries = [(mean(column), stdev(column), len(column)) for column in
zip(*dataset)]
        del(summaries[-1])
        return summaries

# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
        separated = separate_by_class(dataset)
        summaries = dict()
        for class_value, rows in separated.items():
                summaries[class_value] = summarize_dataset(rows)
        return summaries

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
        exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
        return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
        total_rows = sum([summaries[label][0][2] for label in summaries])
        probabilities = dict()
        for class_value, class_summaries in summaries.items():
                probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
                for i in range(len(class_summaries)):
                        mean, stdev, _ = class_summaries[i]
```

```
                        probabilities[class_value] *= calculate_probability(row[i], mean,
stdev)
        return probabilities

# Predict the class for a given row
def predict(summaries, row):
        probabilities = calculate_class_probabilities(summaries, row)
        best_label, best_prob = None, -1
        for class_value, probability in probabilities.items():
                if best_label is None or probability > best_prob:
                        best_prob = probability
                        best_label = class_value
        return best_label

# Make a prediction with Naive Bayes on Iris Dataset
filename = 'iris.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
        str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# fit model
model = summarize_by_class(dataset)
# define a new record
row = [5.7,2.9,4.2,1.3]
# predict the label
label = predict(model, row)
print('Data=%s, Predicted: %s' % (row, label))
```

OUTPUT

```
1
[7.939820817, 0.791637231, 1]

In [9]: runfile('C:/Users/VASUKI/naivebayes.py', wdir='C:/Users/VASUKI')
[Iris-virginica] => 0
[Iris-setosa] => 1
[Iris-versicolor] => 2
Data=[5.7, 2.9, 4.2, 1.3], Predicted: 2

In [10]:
```

Result:

Thus the python program to implement Naïve Bayes Model was executed and output was verified successfully.

Ex.No:4 **Implementation of Bayesian Networks**
Date:

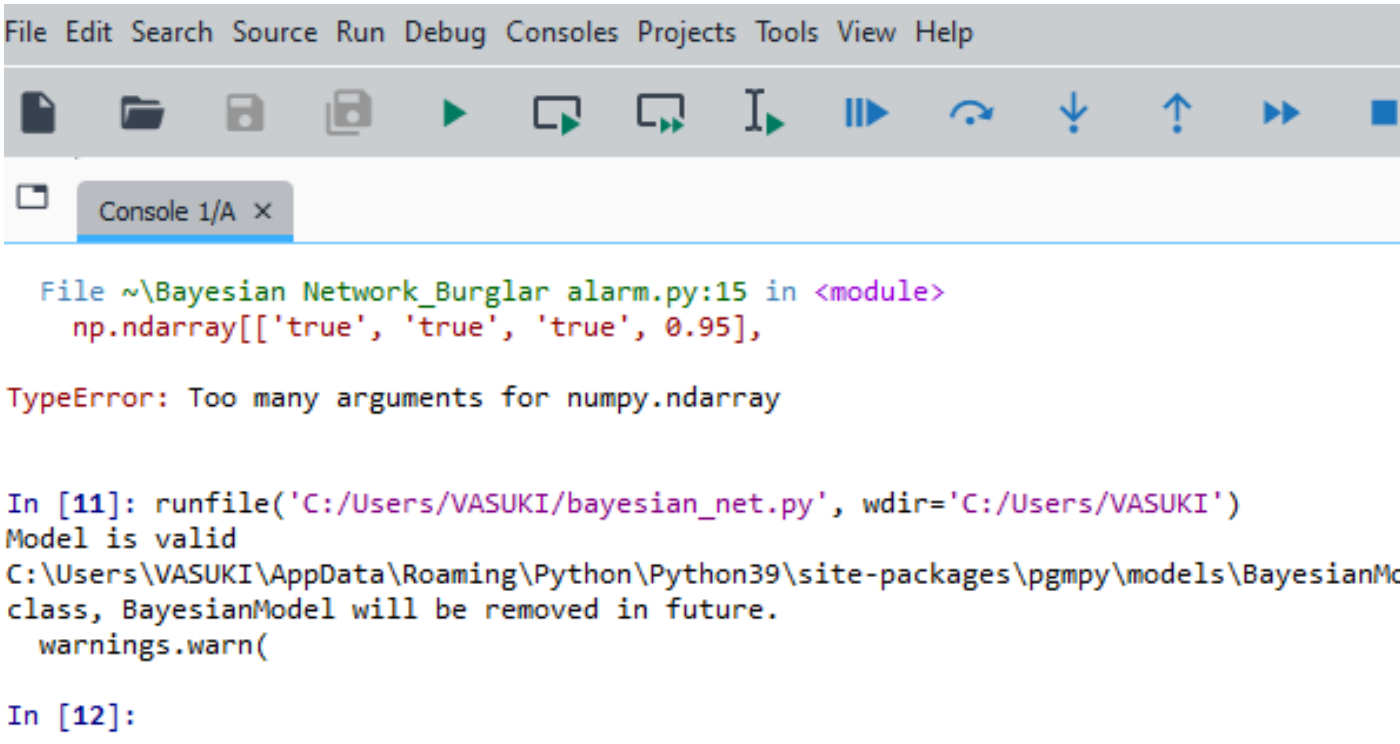
Aim:

To write a python program to implement Bayesian Networks.

Program:

```
import numpy as np
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
# Define the model structure
model = BayesianModel([('Burglary', 'Alarm'), ('Earthquake', 'Alarm'), ('Alarm', 'JohnCalls'),
('Alarm', 'MaryCalls')])
# Define the conditional probability distributions
cpd_burglary = TabularCPD(variable='Burglary',variable_card=2,values=[[0.50],[0.50]])
cpd_earthquake = TabularCPD(variable='Earthquake', variable_card=2, values=[[0.998],
[0.002]])
cpd_alarm = TabularCPD(variable='Alarm', variable_card=2,
                       evidence=['Burglary', 'Earthquake'],
                       evidence_card=[2, 2],
                       values=[[0.999, 0.71, 0.06, 0.05],
                               [0.001, 0.29, 0.94, 0.95]])
cpd_john_calls = TabularCPD(variable='JohnCalls', variable_card=2,
                          evidence=['Alarm'], evidence_card=[2],
                          values=[[0.95, 0.1], [0.05, 0.9]])
cpd_mary_calls = TabularCPD(variable='MaryCalls', variable_card=2,
                          evidence=['Alarm'], evidence_card=[2],
                          values=[[0.99, 0.3], [0.01, 0.7]])
# Add the conditional probability distributions to the model
model.add_cpds(cpd_burglary, cpd_earthquake, cpd_alarm, cpd_john_calls, cpd_mary_calls)
# Check if the model is valid
if model.check_model():
   print("Model is valid")
else:
   print("Model is not valid")
```

OUTPUT



Result:

Thus the python program to implement Bayesian Networks was executed and output was verified successfully.

Ex.No:5 **Build Regression Models**
Date:

Aim:

To write a python program to implement Building RegressionModels.

Program:

a) Simple Linear Regression

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
            marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
```

```
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

b) Multiple Linear Regression

Program:

```
import numpy as np
import pandas as pd
import statistics
import math
from matplotlib import pyplot as plt
import statsmodels.formula.api as smf
import requests # Module to process http/https requests
remote_url="http://54.243.252.9/engr-1330-webroot/8-Labs/Lab29/heart.data.csv
rget = requests.get(remote_url, allow_redirects=True)
open('heart.data.csv','wb').write(rget.content);
heartattack = pd.read_csv('heart.data.csv')
data =
heartattack.rename(columns={"biking":"Bike","smoking":"Smoke","heart.disease":"Disease"
})
print(data.head(3))
# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('Disease ~ Bike + Smoke', data=data)
model = model.fit()
#print(model.summary())
# dir(model) # activate to find attributes
intercept = model.params[0]
slope = model.params[1]
Rsquare = model.rsquared
RMSE = math.sqrt(model.mse_total)
# Predict values
```

```
heartfail = model.predict()

titleline = 'Disease Index versus Lifestyle Variables \n'  + 'R squared = ' +
str(round(Rsquare,3)) + ' \n RMSE = ' + str(round(RMSE,2))
# Plot regression against actual data - What do we see?
plt.figure(figsize=(12, 6))
plt.plot(data['Bike'], data['Disease'], 'o')           # scatter plot showing actual data
plt.plot(data['Bike'], heartfail, marker = 's' ,color ='r', linewidth=0)   # regression line
plt.xlabel('Biking (miles/week)')
plt.ylabel('Disease Index (Admissions/100,000 as per MMWR)')
plt.legend(['Observations','Model Prediction'])
plt.title(titleline)
plt.show()
titleline = 'Disease Index versus Lifestyle Variables \n'  + 'R squared = ' +
str(round(Rsquare,3)) + ' \n RMSE = ' + str(round(RMSE,2))
# Plot regression against actual data - What do we see?
plt.figure(figsize=(12, 6))
plt.plot(data['Smoke'], data['Disease'], 'o')          # scatter plot showing actual data
plt.plot(data['Smoke'], heartfail, marker = 's' ,color ='r', linewidth=0)   # regression line
plt.xlabel('Smoking (packs/week)')
plt.ylabel('Disease Index (Admissions/100,000 as per MMWR)')
plt.legend(['Observations','Model Prediction'])
plt.title(titleline)
plt.show()
print(model.summary())
```

c)Logistic Regression

Program:

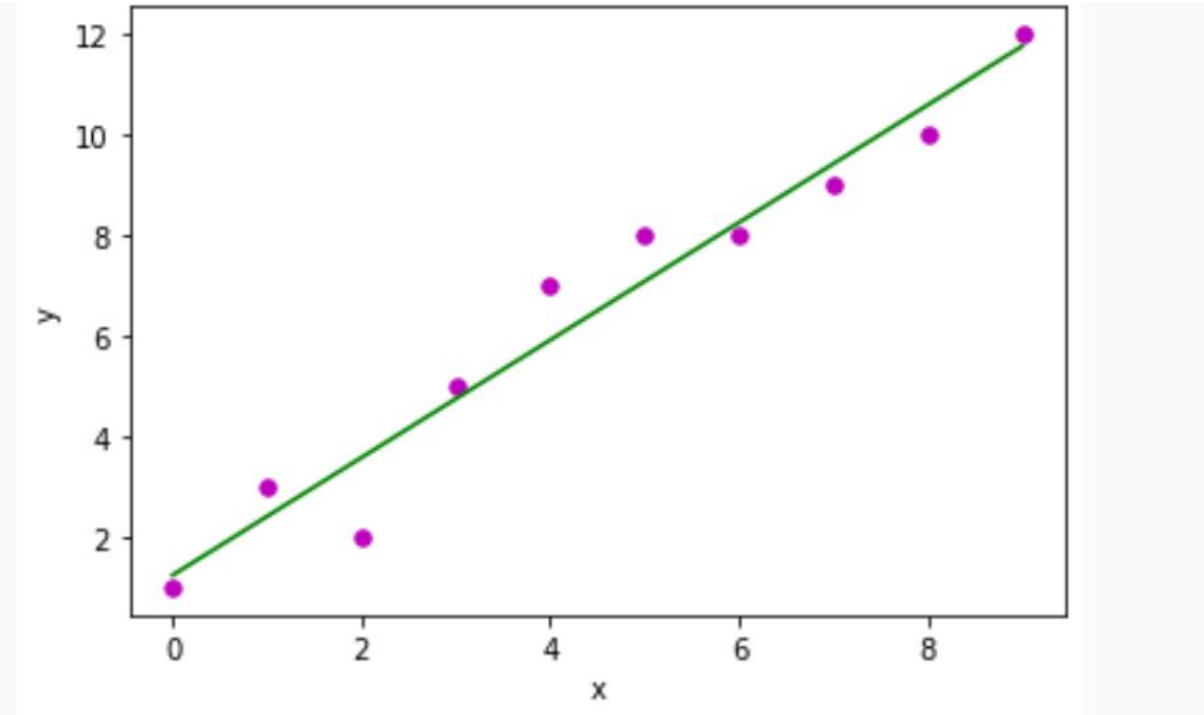
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap
#importing datasets
data_set= pd.read_csv("c:/users/vasuki/carnv.csv")
print(data_set)
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
print("car sales 2nd and 3rd  column")
print(x)
print(y)
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
print("classifying with logistic regression")
print(classifier)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
y_pred= classifier.predict(x_test)
print("selected with given")
print(y_pred)
cm=confusion_matrix(y_test,y_pred)
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('green','yellow' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('green','yellow')))

mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```

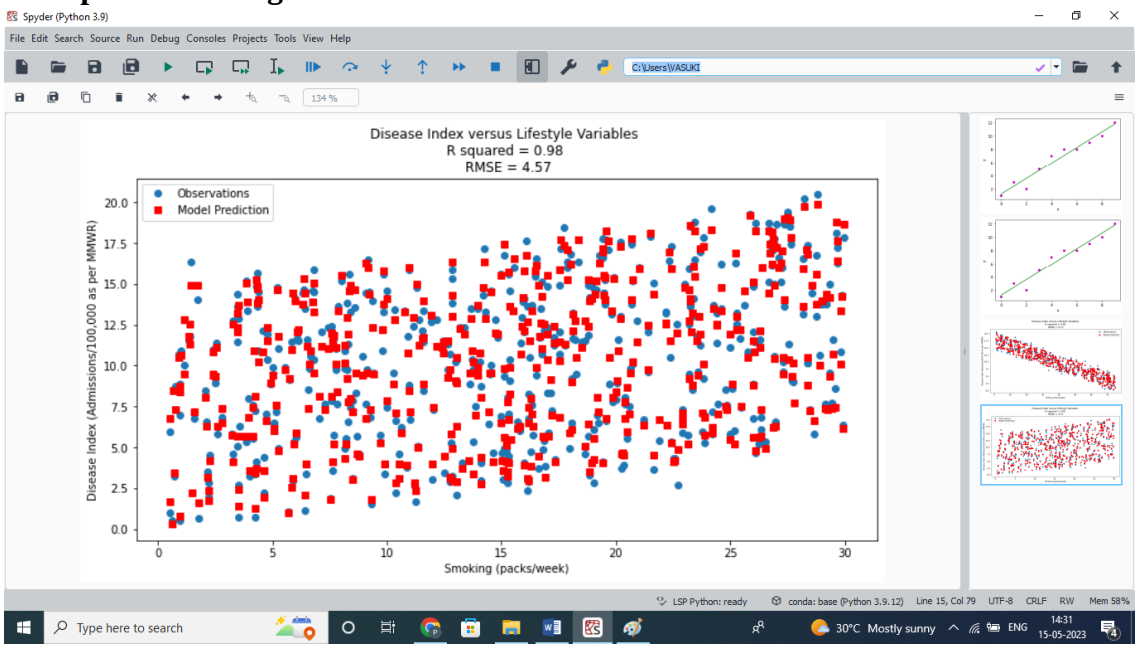
```
    c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT

a. Linear Regression

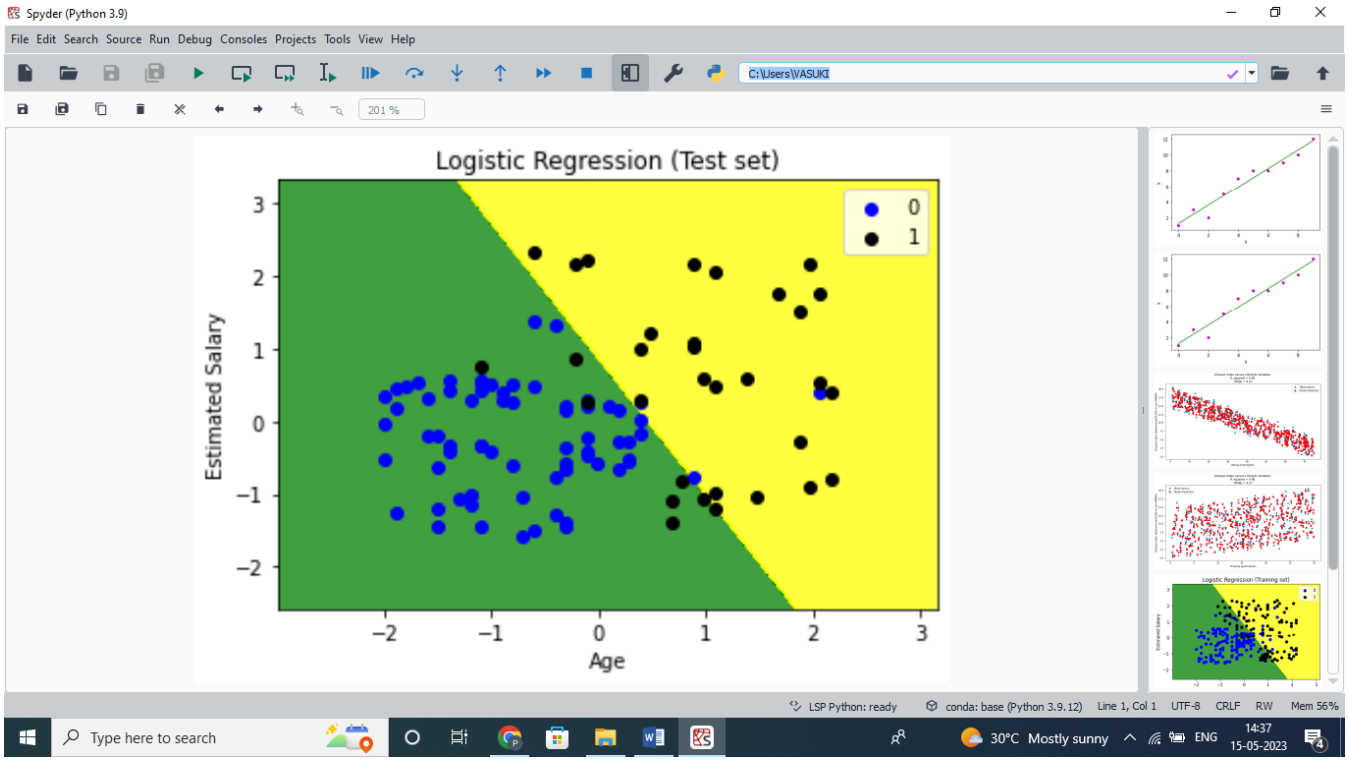


b. Multiple Linear Regression



```
In [18]: runfile('C:/Users/VASUKI/untitled2.py', wdir='C:/Users/VASUKI')
   Unnamed: 0       Bike      Smoke    Disease
0           1  30.801246  10.896608  11.769423
1           2  65.129215   2.219563   2.854081
2           3   1.959665  17.588331  17.177803
                            OLS Regression Results
==============================================================================
Dep. Variable:                Disease   R-squared:                       0.980
Model:                            OLS   Adj. R-squared:                  0.980
Method:                 Least Squares   F-statistic:                 1.190e+04
Date:                Mon, 15 May 2023   Prob (F-statistic):               0.00
Time:                        14:27:09   Log-Likelihood:                -493.68
No. Observations:                 498   AIC:                             993.4
Df Residuals:                     495   BIC:                             1006.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     14.9847      0.080    186.988      0.000      14.827      15.142
Bike          -0.2001      0.001   -146.525      0.000      -0.203      -0.197
Smoke          0.1783      0.004     50.387      0.000       0.171       0.185
==============================================================================
Omnibus:                        2.794   Durbin-Watson:                   1.917
Prob(Omnibus):                  0.247   Jarque-Bera (JB):                2.582
Skew:                          -0.141   Prob(JB):                        0.275
Kurtosis:                       3.211   Cond. No.                         125.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [19]:
```

C. Logistic Regression



Result:

Thus the python program to implement Building Regression Models was executed and output was verified successfully.

Ex.No:6 **Build Decision Tress and Random Forests**
Date:

Aim:

To write a python program to implement building Decision Tress and Random Forests.

Program:

a)Decision Trees

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import ListedColormap
#importing datasets
data_set= pd.read_csv('user_data.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting the dataset into training and test set.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
 #feature Scaling
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix
```

```
cm= confusion_matrix(y_test, y_pred)

x_set, y_set = x_train, y_train

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

fori, j in enumerate(nm.unique(y_set)):

mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

    c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

#Visulaizing the test set result

x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

fori, j in enumerate(nm.unique(y_set)):

mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

    c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm(Test set)')
```

```
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

b)Random Forests

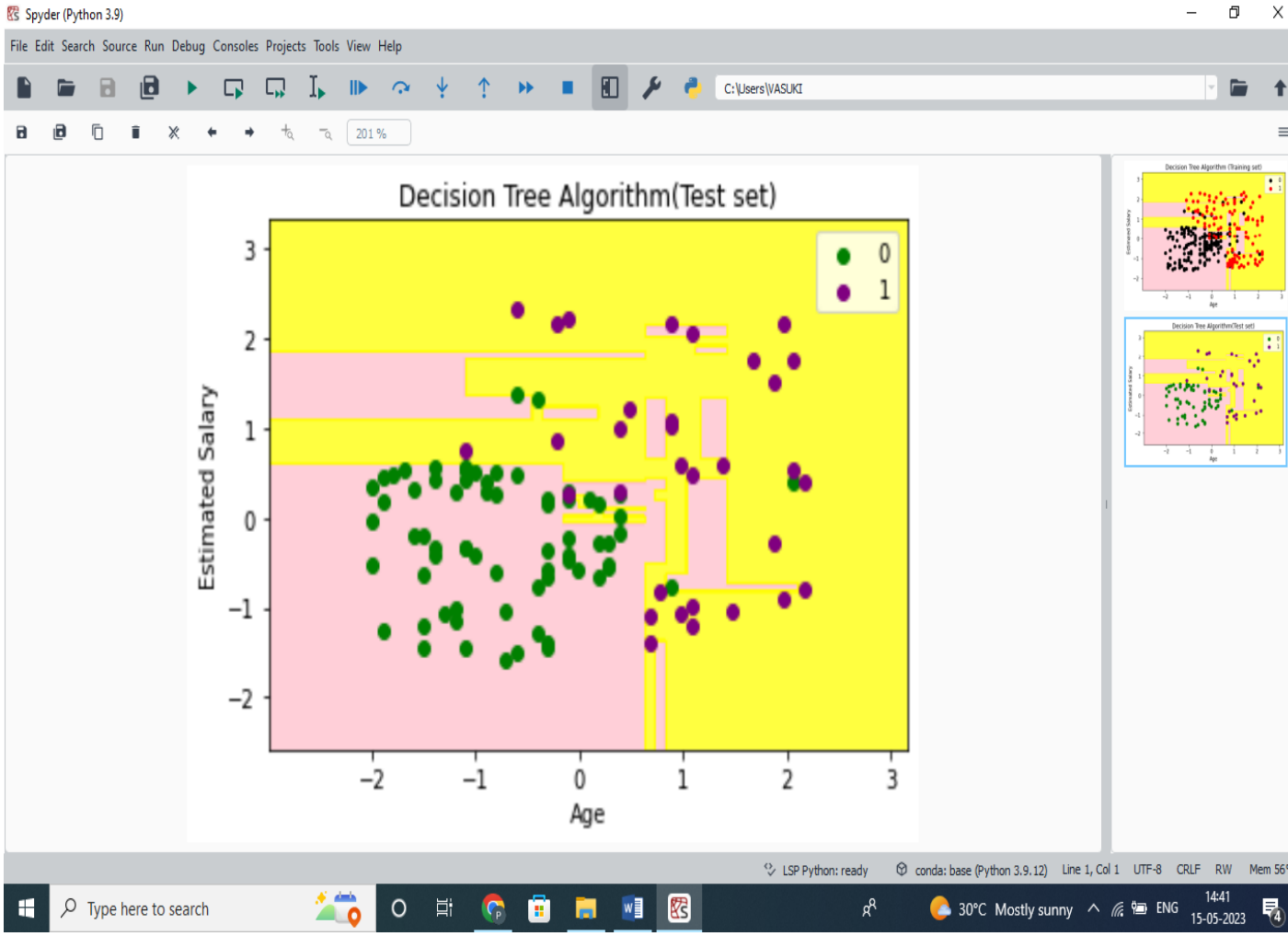
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
 from matplotlib.colors import ListedColormap
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
 from sklearn.metrics import confusion_matrix

#importing datasets
data_set= pd.read_csv('user_data.csv')
 #Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
#Fitting Decision Tree classifier to the training set
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
cm= confusion_matrix(y_test, y_pred)
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()


#Visulaizing the test set result
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
```

```
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT

A. Decision Tree


Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\VASUKI
Decision Tree Algorithm(Test set)
Estimated Salary
Age
0
1
LSP Python: ready
conda: base (Python 3.9.12)
Line 1, Col 1
UTF-8
CRLF
RW
Type here to search
30°C Mostly sunny
ENG
14:41
15-05-2023

B. Random Forest

Result:

Thus the python program to implement building Decision Tress and Random Forests was executed and output was verified successfully.

Ex.No:7 **Build SVM Models**
Date:

Aim:

To write a python program to implement Building SVM Models**.**

Program:

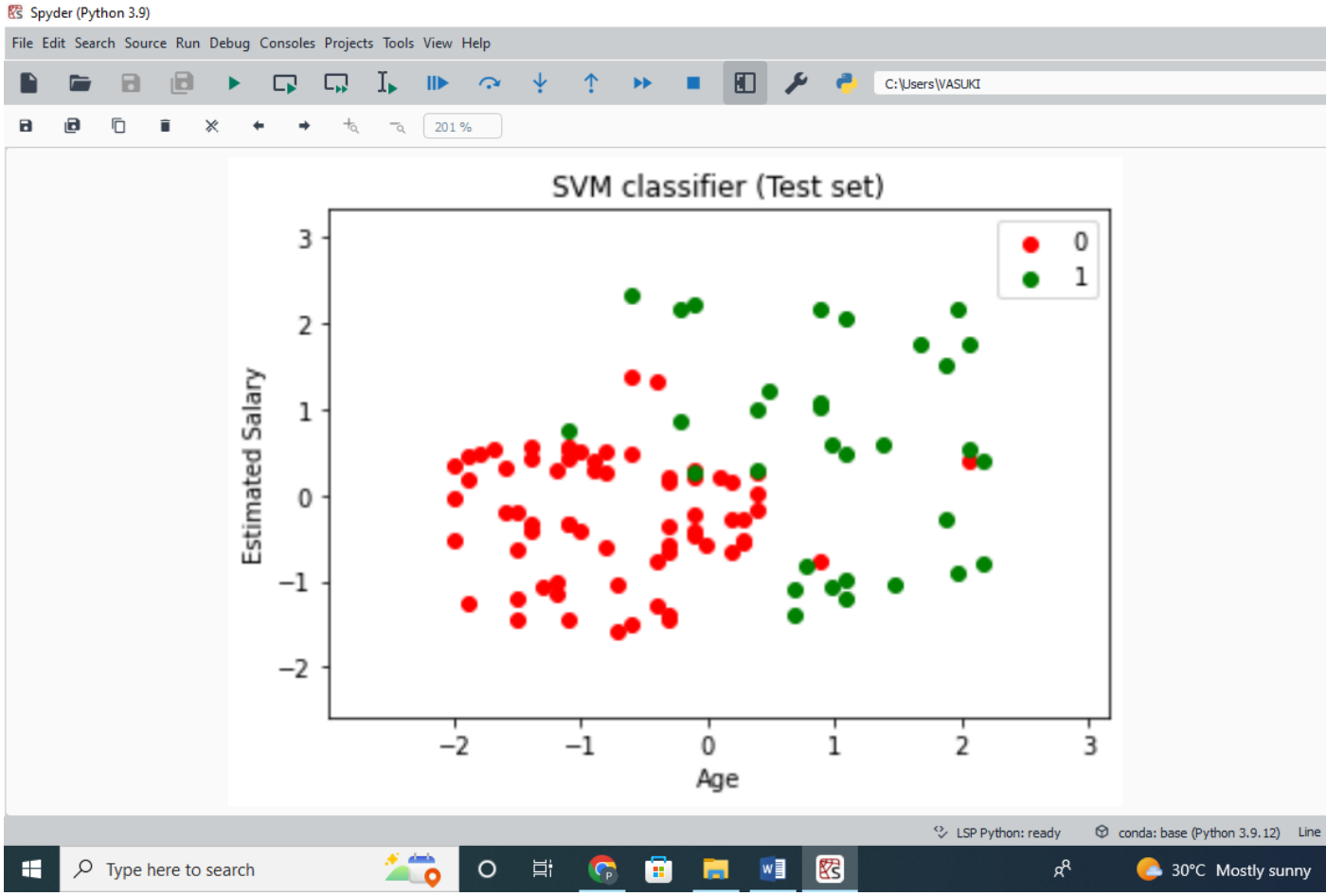
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.svm import SVC # "Support vector classifier"
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap
#importing datasets
data_set= pd.read_csv("c:/users/vasuki/carnv.csv")
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
  # Splitting the dataset into training and test set.
 x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
 cm= confusion_matrix(y_test, y_pred)
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),
```

```
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y_set)):

   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

      c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Training set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()



x_set, y_set = x_test, y_test

x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1,
step  =0.01),

nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y_set)):

   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

      c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Test set)')

mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')
```

mtp.legend()

mtp.show()

OUTPUT



Result:

Thus the python program to implement building SVM Models was executed and output was verified successfully.

Ex.No:8 **Implementation of Ensembling Techniques**
Date:

Aim:

To write a python program to implement Ensembling Techniques.

Program:

a) Voting

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
# loading train data set in dataframe from train_data.csv file
data_set = pd.read_csv("c:/users/vasuki/carnv.csv")
 # getting target data from the dataframe
print(data_set.head())
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
   x,y,test_size=0.20)
# define base models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC()
# define the ensemble model
ensemble = VotingClassifier(estimators=[('lr', model1), ('dt', model2), ('svm', model3)],
voting='hard')
# fit the ensemble model
ensemble.fit(X_train, y_train)
```

```
# make predictions using the ensemble model
y_pred = ensemble.predict(X_test)
print(y_pred)
```

b)Bagging

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
#loading train data set in dataframe from train_data.csv file
data_set = pd.read_csv("c:/users/vasuki/carnv.csv")
 # getting target data from the dataframe
print(data_set.head())
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
    x,y,test_size=0.20)
# define base model
base_model = DecisionTreeClassifier()
# define the bagging model
bagging = BaggingClassifier(base_estimator=base_model, n_estimators=10,
random_state=42)
# fit the bagging model
bagging.fit(X_train, y_train)
# make predictions using the bagging model
y_pred = bagging.predict(X_test)
print(y_pred)
```

c)Boosting

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

data_set = pd.read_csv("c:/users/vasuki/carnv.csv")

# getting target data from the dataframe
print(data_set.head())
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# getting train data from the dataframe
# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
    x,y,test_size=0.20)
# define base model
base_model = DecisionTreeClassifier(max_depth=1)

# define the boosting model
boosting = AdaBoostClassifier(base_estimator=base_model, n_estimators=50,
learning_rate=0.1)

# fit the boosting model
boosting.fit(X_train, y_train)
# make predictions using the boosting model
y_pred = boosting.predict(X_test)
print(y_pred)
```

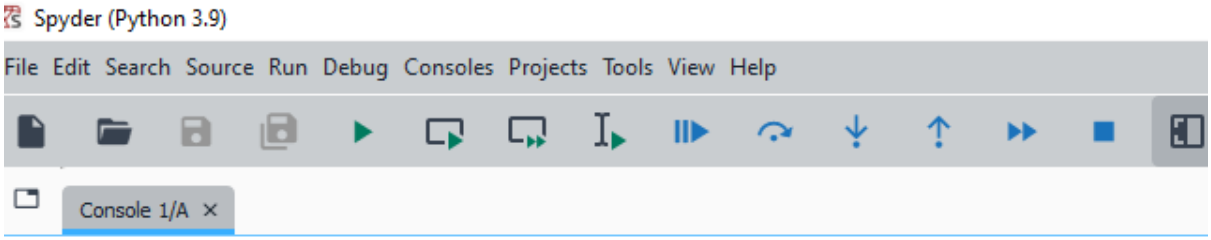
d)Stacking

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

data_set = pd.read_csv("c:/users/vasuki/carnv.csv")

# getting train data from the dataframe
print(data_set.head())
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting between train data into training and validation dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(
   x,y,test_size=0.20)
# define base models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier()
model3 = SVC()

# define the meta-model
meta_model = LogisticRegression()

# define the stacking model
stacking = StackingClassifier(estimators=[('lr', model1), ('dt', model2), ('svm', model3)],
final_estimator=meta_model)

# fit the stacking model
stacking.fit(X_train, y_train)

# make predictions using the stacking model
y_pred = stacking.predict(X_test)
print(y_pred)
```

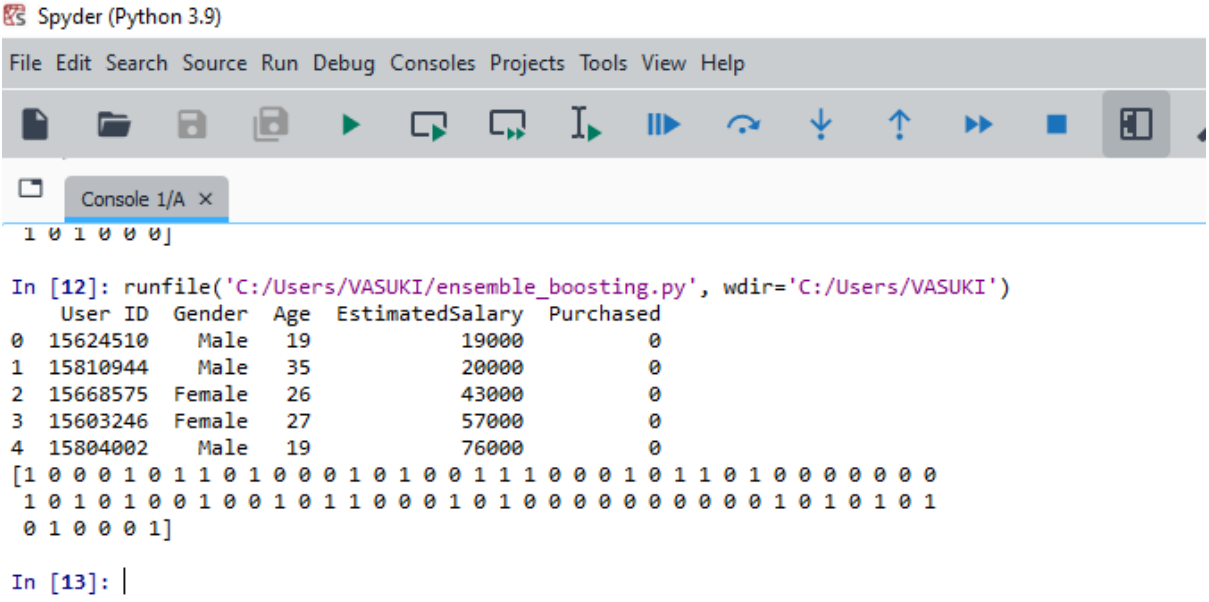
OUTPUT:

A. Bagging:

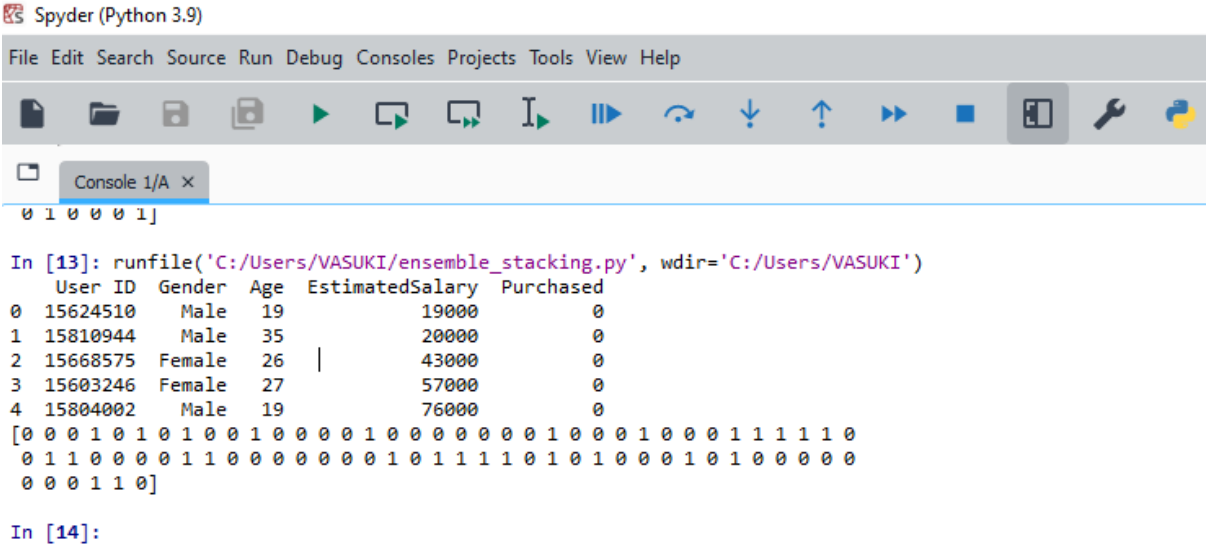



```
In [11]: runfile('C:/Users/VASUKI/ensemble_bagging.py', wdir='C:/Users/VASUKI')
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
[0 0 1 1 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0
 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1 0
 1 0 1 0 0 0]

In [12]:
```

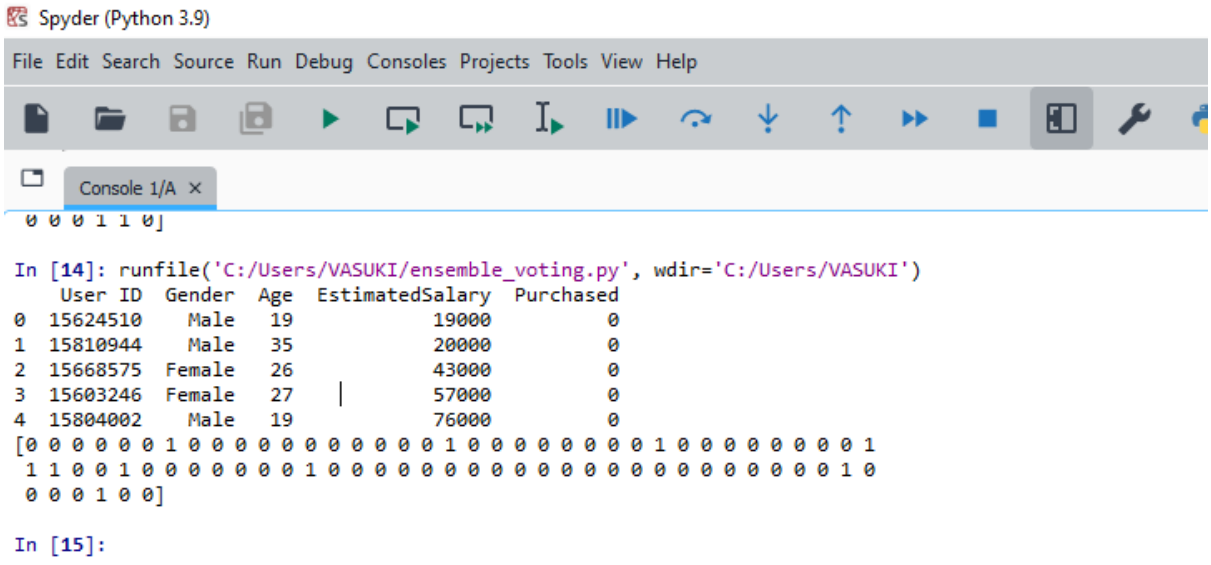
B. Boosting



C. Stacking



D. Voting



Result:

Thus the python program to implement Ensembling Techniques was executed and output was verified successfully.

Ex.No:9 **Implementation of K-means Clustering Algorithm**
Date:

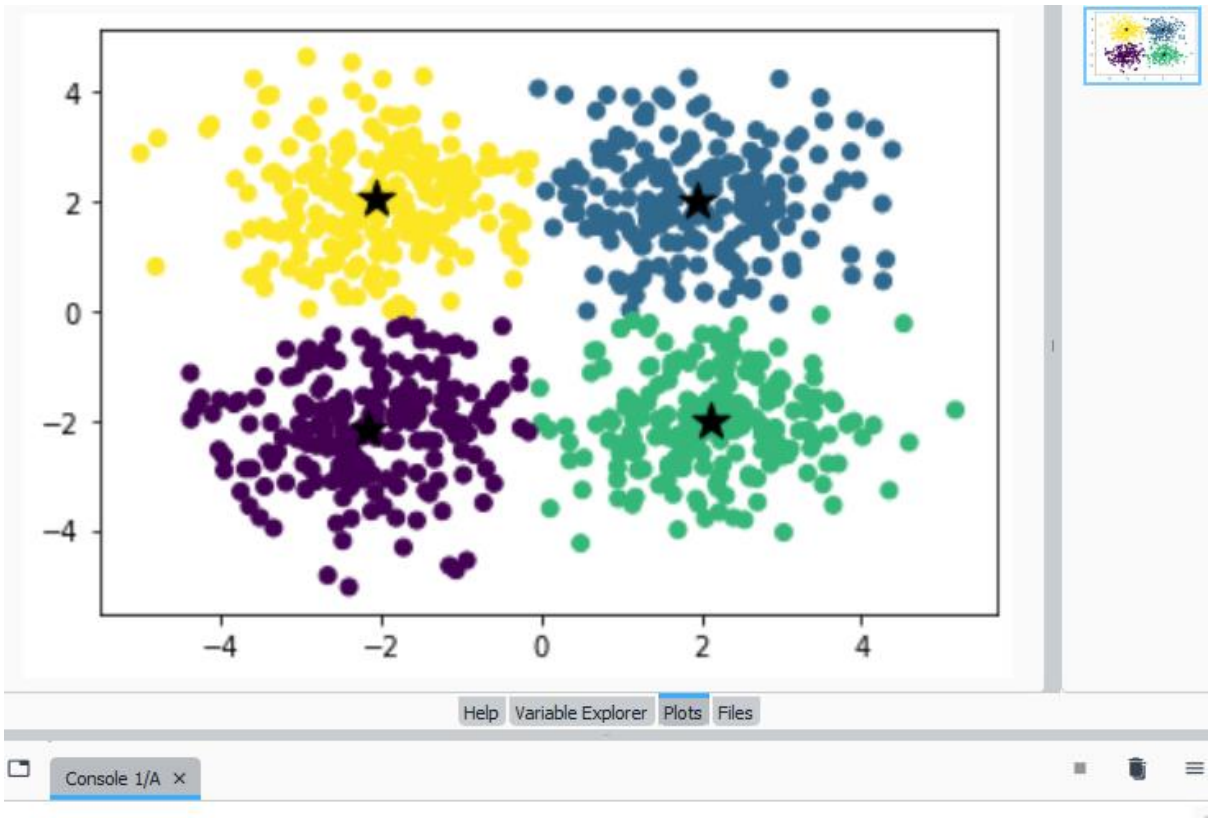
Aim:

To write a python program to implement Clustering Algorithms**.**

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
np.random.seed(0)
X = np.random.randn(200, 2) + np.array([2, 2])
X = np.vstack((X, np.random.randn(200, 2) + np.array([-2, -2])))
X = np.vstack((X, np.random.randn(200, 2) + np.array([2, -2])))
X = np.vstack((X, np.random.randn(200, 2) + np.array([-2, 2])))
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='*', s=200, color='black')
plt.show()
```

OUTPUT



Result:

Thus the python program to implement K-Means Clustering Algorithms was executed and output was verified successfully.

Ex.No:10 **Implementation of EM for Bayesian Networks**
Date:

Aim:

To write a python program to implement EM model for Bayesian Networks.

Program:

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDise = pd.read_csv("c:/users/vasuki/heartdis.csv")
heartDise = heartDise.replace('?',np.nan)
print('Sample instances from the dataset are given below')
print(heartDise.head())
print('\n Attributes and datatypes')
print(heartDise.dtypes)
model=
BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','hear
tdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDise, estimator=MaximumLikelihoodEstimator)
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

OUTPUT:

```
In[2] : runfile('C:/Users/VASUKI/emmodel_bayesiannet.py', wdir='C:/Users/VASUKI')
Sample instances from the dataset are given below

age  gender  cp  trestbps  chol  ...  oldpeak  slope  ca  thal  heartdisease

0   63      1   1      145   233  ...     2.3     3   0    6          0

1   67      1   4      160   286  ...     1.5     2   3    3          2

2   67      1   4      120   229  ...     2.6     2   2    7          1

3   37      1   3      130   250  ...     3.5     3   0    3          0

4   41      0   2      130   204  ...     1.4     1   0    3          0

[5 rows x 14 columns]

 Attributes and datatypes

age             int64

gender          int64

cp              int64

trestbps        int64

chol            int64

fbs             int64

restecg         int64

thalach         int64

exang           int64

oldpeak         float64

slope           int64

ca              object

thal            object

heartdisease    int64

dtype: object

Learning CPD using Maximum likelihood estimators

C:\Users\VASUKI\AppData\Roaming\Python\Python39\site-
packages\pgmpy\models\BayesianModel.py:8: FutureWarning: BayesianModel has been renamed
to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.

  warnings.warn(

 Inferencing with Bayesian Network:
```

1. Probability of HeartDisease given evidence= restecg

heartdisease	phi(heartdisease)
heartdisease(0)	0.1012
heartdisease(1)	0.0000
heartdisease(2)	0.2392
heartdisease(3)	0.2015
heartdisease(4)	0.4581

2. Probability of HeartDisease given evidence= cp

heartdisease	phi(heartdisease)
heartdisease(0)	0.3610
heartdisease(1)	0.2159
heartdisease(2)	0.1373
heartdisease(3)	0.1537
heartdisease(4)	0.1321

Result:

Thus the python program to implement EM Model for Bayesisan Networks was executed and output was verified successfully.

Ex.No:11 **Implementation of a Simple Neural Network**
Date:

Aim:

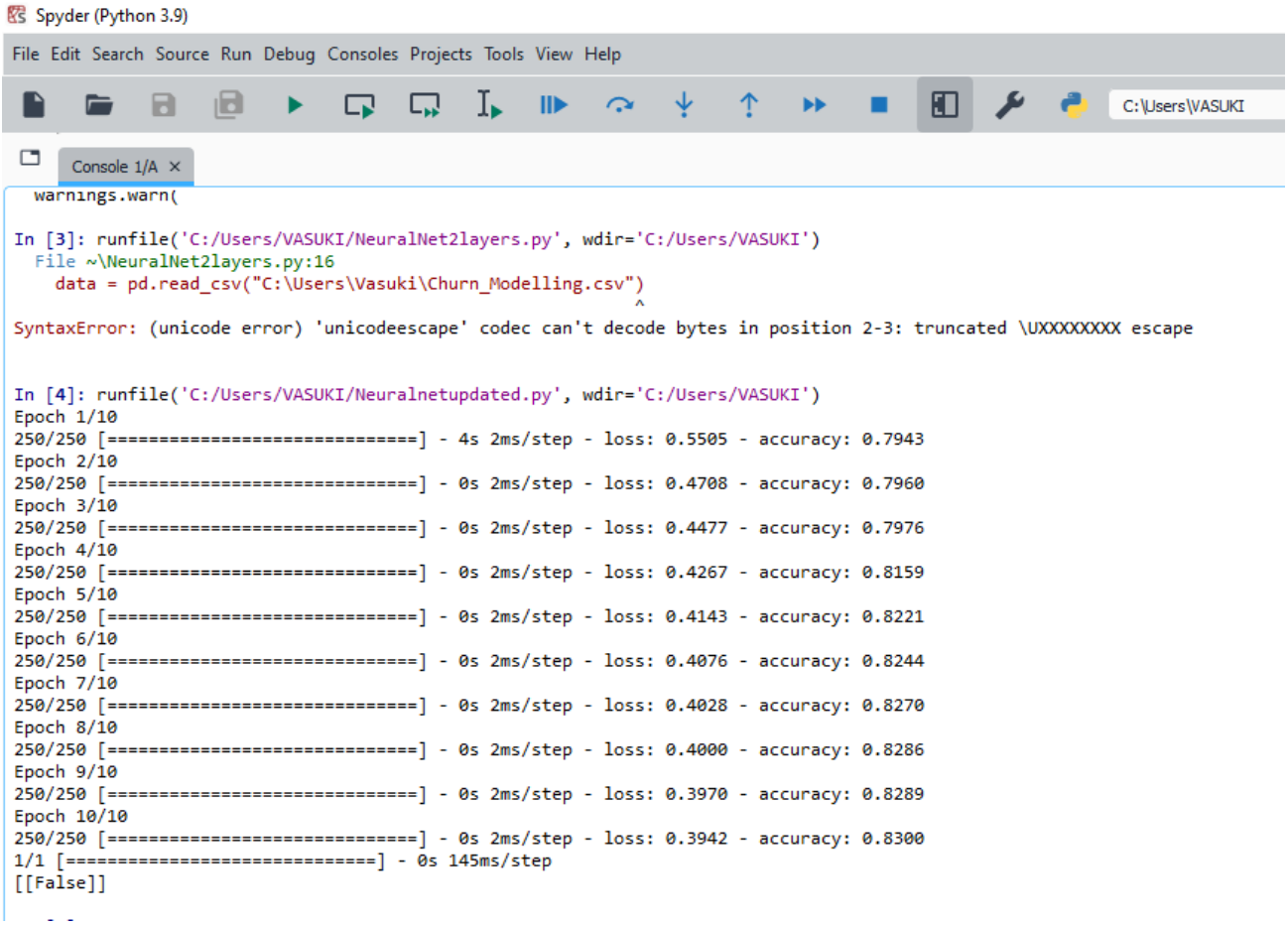
To write a python program to implement a Simple Neural Networks.

Program

```
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import tensorflow as tf

# Loading dataset
data = pd.read_csv("C:/users/vasuki/Churn.csv")

# Preprocessing data
X = data.iloc[:, 3:-1].values
Y = data.iloc[:, -1].values

LE1 = LabelEncoder()
X[:,2] = LE1.fit_transform(X[:,2])
ct
=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[1])],remainder="passthrou
gh")
X = ct.fit_transform(X)

# Splitting dataset into training and testing dataset
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the neural network model
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=6, activation="relu"))
ann.add(tf.keras.layers.Dense(units=6, activation="relu"))
ann.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Compiling the model
ann.compile(optimizer="adam", loss="binary_crossentropy", metrics=['accuracy'])
```

```
# Training the model
ann.fit(X_train, Y_train, batch_size=32, epochs=10)

# Predicting a single observation
single_prediction = ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]]))
> 0.5
print(single_prediction)

# Saving the model
```

OUTPUT:



Result:

Thus the python program to implement to implement a Simple Neural Networks was executed and output was verified successfully.

Ex.No:12 **Implementation of a Deep Neural Network**
Date:

Aim:

To write a python program to implement a Deep Neural Networks.

Program:

```
from tensorflow.keras import models, layers, utils, backend as K
import matplotlib.pyplot as plt
import os
import numpy as np
import shap
import tensorflow as tf


model = models.Sequential(name="Perceptron", layers=[
   layers.Dense(            #a fully connected layer
        name="dense",
        input_dim=3,       #with 3 features as the input
        units=1,           #and 1 node because we want 1 output
        activation='linear' #f(x)=x
   )
])

model.summary()
def binary_step_activation(x):

   return K.switch(x>0, tf.math.divide(x,x), tf.math.multiply(x,0))

# build the model
model = models.Sequential(name="Perceptron", layers=[
     layers.Dense(
        name="dense",
        input_dim=3,
        units=1,
        activation=binary_step_activation
     )
])

n_features = 10
model = models.Sequential(name="DeepNN", layers=[
   layers.Dense(name="h1", input_dim=n_features,
               units=int(round((n_features+1)/2)),
               activation='relu'),
```

```
    layers.Dropout(name="drop1", rate=0.2),


    layers.Dense(name="h2", units=int(round((n_features+1)/4)),
                 activation='relu'),
    layers.Dropout(name="drop2", rate=0.2),


    layers.Dense(name="output", units=1, activation='sigmoid')
])
model.summary()

# Perceptron
inputs = layers.Input(name="input", shape=(3,))
outputs = layers.Dense(name="output", units=1,
                       activation='linear')(inputs)
model = models.Model(inputs=inputs, outputs=outputs,
                     name="Perceptron")


inputs = layers.Input(name="input", shape=(n_features,))

h1 = layers.Dense(name="h1", units=int(round((n_features+1)/2)), activation='relu')(inputs)
h1 = layers.Dropout(name="drop1", rate=0.2)(h1)

h2 = layers.Dense(name="h2", units=int(round((n_features+1)/4)), activation='relu')(h1)
h2 = layers.Dropout(name="drop2", rate=0.2)(h2)

outputs = layers.Dense(name="output", units=1, activation='sigmoid')(h2)
model = models.Model(inputs=inputs, outputs=outputs, name="DeepNN")
print("Model with sigmoid func")
model.summary()

def utils_nn_config(model):
    lst_layers = []
    if "Sequential" in str(model):  # Sequential doesn't show the input layer
        layer = model.layers[0]
        lst_layers.append({"name": "input", "in": int(layer.input.shape[-1]), "neurons": 0,
                           "out": int(layer.input.shape[-1]), "activation": None,
                           "params": 0, "bias": 0})
    for layer in model.layers:
        try:
            dic_layer = {"name": layer.name, "in": int(layer.input.shape[-1]), "neurons":
layer.units,
                         "out": int(layer.output.shape[-1]), "activation":
layer.get_config()["activation"],
```

```
                "params": layer.get_weights()[0], "bias": layer.get_weights()[1]}
        except:
            dic_layer = {"name": layer.name, "in": int(layer.input.shape[-1]), "neurons": 0,
                "out": int(layer.output.shape[-1]), "activation": None,
                "params": 0, "bias": 0}
        lst_layers.append(dic_layer)
    return lst_layers


def visualize_nn(model, description=False, figsize=(10, 8)):
    # get layers info
    lst_layers = utils_nn_config(model)
    layer_sizes = [layer["out"] for layer in lst_layers]

    # fig setup
    fig = plt.figure(figsize=figsize)
    ax = fig.gca()
    ax.set(title=model.name)
    ax.axis('off')
    left, right, bottom, top = 0.1, 0.9, 0.1, 0.9
    x_space = (right - left) / float(len(layer_sizes) - 1)
    y_space = (top - bottom) / float(max(layer_sizes))
    p = 0.025

    # nodes
    for i, n in enumerate(layer_sizes):
        top_on_layer = y_space * (n - 1) / 2.0 + (top + bottom) / 2.0
        layer = lst_layers[i]
        color = "green" if i in [0, len(layer_sizes) - 1] else "blue"
        color = "red" if (layer['neurons'] == 0) and (i > 0) else color

        # add description
        if description is True:
            d = i if i == 0 else i - 0.5
            if layer['activation'] is None:
                plt.text(x=left + d * x_space, y=top, fontsize=10, color=color,
s=layer["name"].upper())
            else:
                plt.text(x=left + d * x_space, y=top, fontsize=10, color=color,
s=layer["name"].upper())
                plt.text(x=left + d * x_space, y=top - p, fontsize=10, color=color,
s=layer['activation'] + " (")
                plt.text(x=left + d * x_space, y=top - 2 * p, fontsize=10, color=color,
                    s="Σ" + str(layer['in']) + "[X*w]+b")
                out = " Y" if i == len(layer_sizes) - 1 else " out"
                plt.text(x=left + d * x_space, y=top - 3 * p, fontsize=10, color=color,
```

```
                s='') = '' + str(layer['neurons']) + out)

        # circles
        for m in range(n):
            color = "limegreen" if color == "green" else color
            circle = plt.Circle(xy=(left + i * x_space, top_on_layer - m * y_space - 4 * p),
radius=y_space / 4.0,
                                color=color, ec='k', zorder=4)
            ax.add_artist(circle)

            # add text
            if i == 0:
                plt.text(x=left - 4 * p, y=top_on_layer - m * y_space - 4 * p, fontsize=10, s=r'$X_{'
+ str(m + 1) + '}$')
            elif i == len(layer_sizes) - 1:
                plt.text(x=right + 4 * p, y=top_on_layer - m * y_space - 4 * p, fontsize=10,
                    s=r'$y_{' + str(m + 1) + '}$')
            else:
                plt.text(x=left + i * x_space + p, y=top_on_layer - m * y_space + (y_space / 8. +
0.01 * y_space) - 4 * p,
                    fontsize=10, s=r'$H_{' + str(m + 1) + '}$')

    # links
    for i, (n_a, n_b) in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
        layer = lst_layers[i + 1]
        color = "green" if i == len(layer_sizes) - 2 else "blue"
        color = "red" if layer['neurons'] == 0 else color
        layer_top_a = y_space * (n_a - 1) / 2. + (top + bottom) / 2. - 4 * p
        layer_top_b = y_space * (n_b - 1) / 2. + (top + bottom) / 2. - 4 * p
        for m in range(n_a):
            for o in range(n_b):
                line = plt.Line2D([i * x_space + left, (i + 1) * x_space + left],
                                  [layer_top_a - m * y_space, layer_top_b - o * y_space],
                                  c=color, alpha=0.5)
                if layer['activation'] is None:
                    if o == m:
                        ax.add_artist(line)
                else:
                    ax.add_artist(line)
    plt.show()


# define metrics
def Recall(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
```

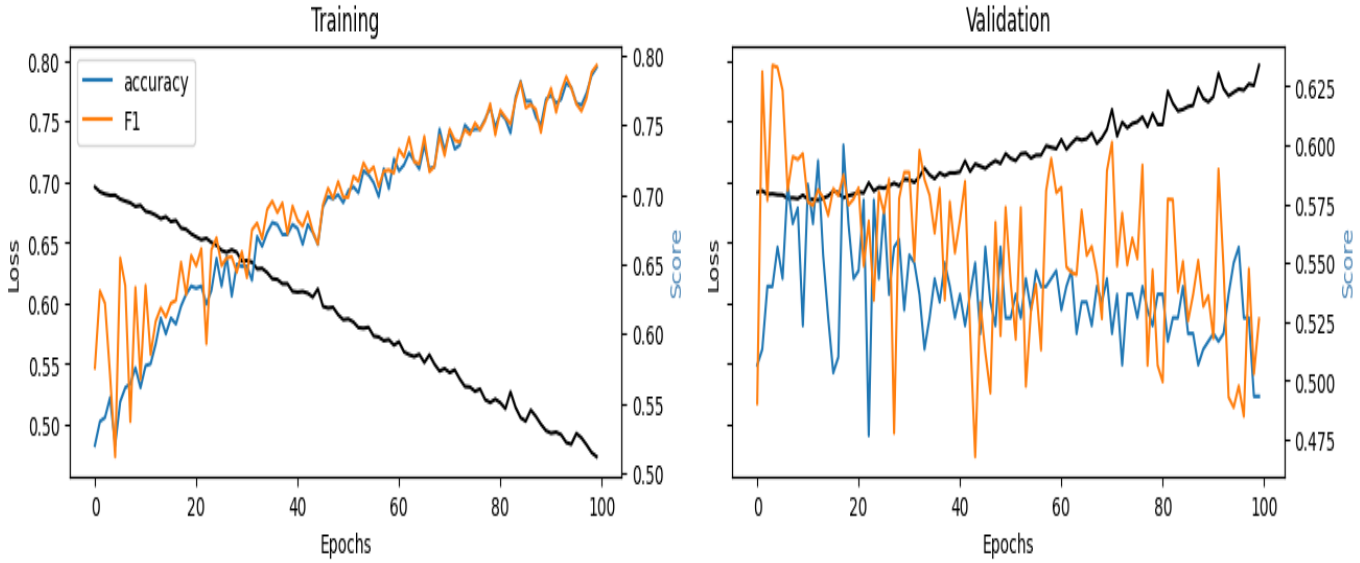
```
    recall = true_positives / (possible_positives + K.epsilon())
    return recall


def Precision(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision


def F1(y_true, y_pred):
    precision = Precision(y_true, y_pred)
    recall = Recall(y_true, y_pred)
    return 2 * ((precision * recall) / (precision + recall + K.epsilon()))


# create the neural network model
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

# compile the neural network
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy', F1])

X = np.random.rand(1000, 10)
y = np.random.choice([1, 0], size=1000)

# train/validation
training = model.fit(x=X, y=y, batch_size=32, epochs=100, shuffle=True, verbose=0,
validation_split=0.3)

# plot
metrics = [k for k in training.history.keys() if ("loss" not in k) and ("val" not in k)]
fig, ax = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(15, 3))

# training
ax[0].set(title="Training")
ax11 = ax[0].twinx()
ax[0].plot(training.history['loss'], color='black')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Loss', color='black')
for metric in metrics:
    ax11.plot(training.history[metric], label=metric)
```

```
ax11.set_ylabel("Score", color='steelblue')
ax11.legend()

# validation
ax[1].set(title="Validation")
ax22 = ax[1].twinx()
ax[1].plot(training.history['val_loss'], color='black')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss', color='black')
for metric in metrics:
   ax22.plot(training.history['val_' + metric], label=metric)
ax22.set_ylabel("Score", color="steelblue")
plt.show()
```

OUTPUT

```
Model: "Perceptron"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
Model: "DeepNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 h1 (Dense)                  (None, 6)                 66

 drop1 (Dropout)             (None, 6)                 0

 h2 (Dense)                  (None, 3)                 21

 drop2 (Dropout)             (None, 3)                 0

 output (Dense)              (None, 1)                 4

=================================================================
Total params: 91
Trainable params: 91
Non-trainable params: 0
_________________________________________________________________
Model with sigmoid func
Model: "DeepNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 10)]              0

 h1 (Dense)                  (None, 6)                 66
```

```
 drop1 (Dropout)              (None, 6)                 0

 h2 (Dense)                   (None, 3)                 21

 drop2 (Dropout)              (None, 3)                 0

 output (Dense)               (None, 1)                 4

=================================================================
Total params: 91
Trainable params: 91
Non-trainable params: 0
```



Result:

Thus the python program to implement to implement a Deep Neural Networks was executed and output was verified successfully.